Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 11 (2013) 400 – 405

www.elsevier.com/locate/procedia

2nd International Through-life Engineering Services Conference

Unicellular Self-Healing Electronic Array

Mohammad Samie[1*], Gabriel Dragffy[2], Tony Pipe[2], Suresh Perinpanayagam[1]

[1]School of Applied Sciences,Cranfield University, UK
[2]Bristol Robotics Labortory, Bristol, UK

* Corresponding author. Tel.: +44-; *E-mail address:* m.samie@cranfield.ac.uk

Abstract

This paper presents on-line fault detection and fault repair capability of our Unitronics architecture, based on a bio-inspired prokaryotic bacterial colony model. At the device programming level, it appears as a cellular FPGA-like system; however, underlying structures transpose it into an inherently self-healing and fault tolerant electronics system. An e-puck object avoidance robot controller was built to demonstrate all the underlying theories of our research. The robot successfully demonstrated that it was able to cope with multiple, simultaneously occurring faults on-line whilst the robot was being controlled to move in a „figure 8"-like manner. Integrity of the system is continuously monitored on-line, and if a fault is detected its location is automatically identified. Detection will trigger an on-line self-repair process. The amount of repair only depends on the number of spare cells the system is equipped with. The embedded fault repair mechanism uses significantly less memory for gene storage and considerably less hardware overall for target system implementation than any previously proposed bio-inspired architecture.

Keywords: Self-Healing; Fault-Detection; Fault-Repair; Cellular electroincs

1. Introduction

The early 90's saw the first attempts [1, 2] to construct bio-inspired electronics systems using a cellular array type architecture. They were based on properties and characteristics of, and used mechanisms found in, multi-cellular eukaryotic organisms. Here, similar to nature, all the cells of the system, in order to configure them for a specific function, contained a full or a partial copy of the organism's DNA (genome). This approach has invariably resulted in a large amount of DNA memory in each cell. The task of the memory is to store the genetic behaviour (DNA) of each cell of the system, in the form of configuration bits (genes) for both its functional characteristic and for the necessary interconnects. Embryonics and the POEtic projects are examples of eukaryotic bio-inspired systems [3, 4]. CellMatrix offers an alternative approach for cellular implementation of systems [5].

Self-healing properties, immunological protection and learning abilities are amongst the advantages offered by the eukaryotic model. All previously proposed Embryonic systems suffer from several disadvantages in silicon area consumption, redundancy, storing large amount of redundant information (each cell required a copy the entire DNA of the system or a large part of it) increases the probability of hardware faults and information mutation in the memory cells.

We suggest that if a model with at least similar performance advantages but based on a simpler form of biological life could be developed, then there is a chance that it might provide a solution to the above problems. We believe that the Unitronic artificial system, which is inspired by primitive unicellular beings called prokaryotes, in particular, bacteria, with its structure and characteristics does indeed offer the answer. It combats the problem of high genome redundancy, thus increases system reliability and is in all respect superior to all Embryonics based systems.

The novel artificial prokaryotic model we have proposed [6, 7] is a solution to build efficient fault tolerant hardware systems. It offers: efficient optimisation of genome

doi:10.1016/j.procir.2013.07.009

redundancy, smaller silicon area, smaller memory for the storage of redundant (back-up) configuration information and requiring less logic support [6]. In our prokaryote model, the cell is only required to store its own configuration bits and some non-configuration bits that support self-repair and not a large part or the entire DNA of the system. Self-repair is achieved by a simple cell elimination process. A new self-test methodology was proposed [8] that offers an acceptable overhead compromise between time and hardware redundancy and guarantees that not only functionality, but all interconnect lines of the cellular system, are also tested.

1.1. Prokaryotic Bio-Inspired Model

The prokaryotic bio-inspired model [6, 7] offers a multi-layer architecture of programmable universal cells. Each cell consists of a function unit (FU), a communication block and a memory block. The latter contains the configuration bits (gene) of the cell that define the required behaviour of both the function unit and that of the communication block, and non-configuration bits which assist self-repair if a fault is detected. Since the task of the gene in the configuration register (CR) is to code the behaviour of a cell so it is termed as a coding gene, while the gene in the non-configuration register (non-CR) that assists self-repair is a non-coding gene. Thus each cell's genome could be viewed as consisting of one coding and one non-coding gene. The non-coding genes are assisting the functionality and the recovery of the coding genes both for the cell in which they reside and for other cells.

In a multi-layered prokaryotic model, cells form clusters, which in turn form colonies and on the top level biofilm communities are formed by colonies. Although the individual bacterial cells' genomes, in a family of species, are the same, due to continual evolution that takes place, mutation will differentiate them. Disregarding these small amounts of differences there will always be a strand in their DNA which they all share and is common to them all. Similarly therefore, in an artificial system family, clusters could be formed with cells that demonstrate similarity in their configuration bits. These cells, although they are unique and different in their own rights, do display similarity through a shared value (Csv) that is common to every cell in a cluster. Characteristics of artificial cells are stored in the form of bits in their configuration register and form its configuration vector (Ccv). Therefore every cells' configuration vector is made up of a value that the cells share (Csv) and is common to them all, and by a differential value (Δg) that distinguishes the cells from another. The configuration vector of a cell can therefore be described by Equation 1.

$$Ccv = Csv + \Delta g \quad (1)$$

or generally as:

$$Ccv = f(Csv, \Delta g)$$

where f in refers to the evolutionary function and in the simplest form could be considered as XOR or subtraction functions.

A cluster forms the first community layer. It is a convenient collection of cells to aid self-repair. A cluster is a community of genetically related entities that need not have any functional relationship. In the simplest form, two different types of clusters may be defined: as shared value cluster (sv-cluster), and gene difference value cluster (Δg-cluster). The first one refers to those cells in the colony that have the same shared value of their configuration bits and hence originate from the same species. The second one refers to those cells that have the same genetic difference from their base species. Components of cells and clusters are shown in Fig. 1.

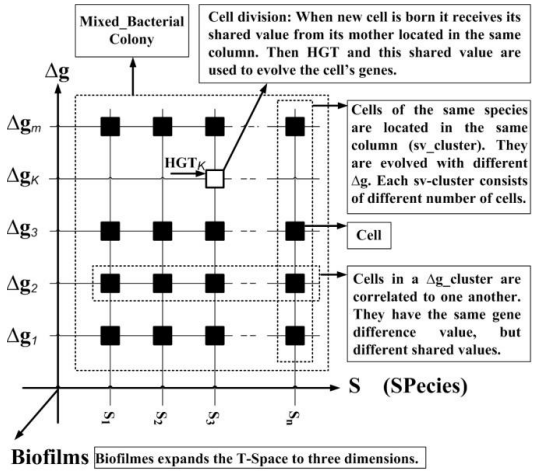

Fig. 1, Prokaryotic Bio-Inspired Model.

1.2. Self-Repair

Cell division requires a 'new' cell that, during the repair process, will be configured the same as the eliminated faulty cell. Since, unlike in nature, our current technology does not facilitate birth of hardware cells, artificial systems must have some redundancy through the availability of spare cells. If a system consists of n available cells of which a specific application uses m cells, then the number of available spare cells is n-m.

Consider that cell k (between cells 1 to m) is detected as faulty. In this case all cells located between k+1 to m are shifted one cell forward to cells k+2 to m+1, where cell m+1 is part of the system's redundant available cells. Cell k+1 will act as a 'spare cell' and will replace the faulty cell. Cell division is a two-step process: Shifting prepares a spare cell adjacent to the faulty one; and, Calculating and loading the shared value of faulty cell into the spare cell.

These will be followed by a differentiation process where, from the shared value the cell's configuration, vector (Ccv) will be evolved. Lack of the shifting process is the only difference between hardware and software fault repair. If several faulty cells simultaneously develop a fault then, following their elimination, the same shifting process will take place and the number of available redundant cells will be accordingly reduced. During shifting, cells are individually checked for integrity and simply by-passed if they were previously killed, while their neighbours will serve as spare cells and will take over the functionality of the faulty ones.

We mentioned previously that clusters are communities of software related cells that have the same shared value, or the same differential parameter. The genome (CGen) of a sv-

cluster is made up as a union (∪) of the genes (g) of its individual cells and can be expressed as:

$$CGen(Tsvi) = \cup g(Tsvi, T\Delta gj), \quad i \in \{1, 2, ..., v\} \ \& \ j \in \{1, 2,..., w\} \tag{2}$$

where j refers to the individual cells in the cluster having the same shared value addressed by Tsvi and i refers to the ith sv-cluster, Tsvi. These clusters are shown by the vertical lines in the Fig. 2. A similar equation can be formulated for Δg-clusters that have the same differential parameters:

$$CGen(T\Delta gj) = \cup g(Tsvi, T\Delta gj), \quad i \in \{1,2, ..., v\} \ \& \ j \in \{1, 2, ..., w\} \tag{3}$$

where i refers to the individual cells in the cluster having the same differential parameters addressed by TΔgj and j refers to the jth Δg-cluster, TΔgj. These clusters are shown by the horizontal lines in Fig. 2. It also shows an example of how the physical placement of a faulty cell in the array differs from its placement in T-Space. Every cell in Fig. 2 has its place both in the sv-cluster and in the Δg-cluster. When faults are detected, for as long as one healthy cell exists in both CGen(Tsvi) and in CGen(TΔgj), the gene of faulty cell can always be recovered with Tsvi and TΔgj. Fig. 2 also shows that cells do not need to be physically sorted when comparing their locations in T-Space.

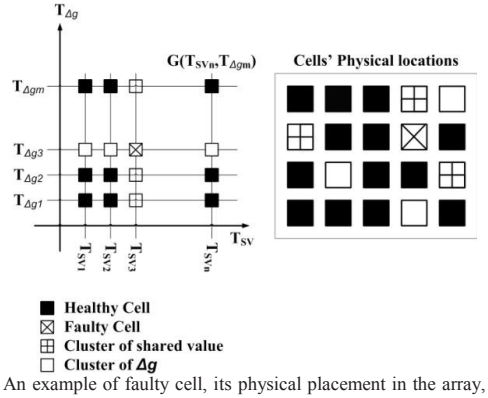

Fig. 2, An example of faulty cell, its physical placement in the array, and in the T-Space.

Equation 2 shows that how, in a prokaryotic model based system, clusters compress the system's genome. Every cell in the appropriate clusters of CGen (Tsv) (vertically sorted in Fig. 2) is expressed with a same shared value and some differential parameters. The self-repair process uses this shared value during cell division by copying that of the faulty cell into the spare cell. It is only the differential parameter (Δg) that distinguishes the cell now from other cells in the cluster. The healthy configuration vector can be recovered by differentiating this shared value with the faulty cell's Δg. It can be extracted from the Δg-cluster of CGen(TΔg) by TΔg, where the faulty cell belonged. Since all cells in a sv-cluster have the same Csv, it is readily available from any of its cells. It is a calculable entity and therefore requires no storage. Finally, the configuration vector of the faulty cell can be calculated as CCVi = CSVi + Δgj. For safety and for easy self-repair purposes neither Δg nor TΔg is saved in the cell's own non-configuration register but another cell will host them. In this way, every cell in the cluster has a back-up memory in the form of a non-configuration register that stores information for other cells. Self-repair process takes place in three steps:

i. Cell division.

ii. Identifying the species of the faulty cell, the sv-cluster and the actual shared value.

iii. Differentiating the shared value with Δg obtained from, Δg-cluster.

Steps 2 and 3 can only be executed if the faulty cell's tags remains healthy. Since the bit requirement of the tags is considerably less than that of Ccv and Δg, this condition is not difficult to meet. However, should the tag values still mutate, additional safety storage is provided by fault tolerant RAMs in an external backup memory.

1.3. Self-Test

The bio-inspired self-test we are proposing is based on two characteristics of biological systems:

• In nature, the DNA is a double helix, a duplicated sequence of complementary genes. It means that both sequences define exactly the same organism with exactly the same features. Therefore one strand is sufficient for the growth and development of an organism [9].

• Transposons (formally termed jumping genes) are sequences of DNA that can move around to a different position within the genome of a single cell. Such mobile genetic elements can move within the genome from one position to another using a "cut and paste" mechanism [10].

These two characteristics found in nature can be used to inspire the development of a bio-inspired self-test model for artificial systems by observing that:

i. If we could guarantee that by configuring the processing elements of an artificial cell with both its gene and complementary gene, their functionality would remain the same and

ii.That using the concept of the jumping genes mechanism could offer a solution to switch over and substitute input signals of such processing elements and interchange their outputs.

1.4. Unitronics Architecture

Embryonics, inspired by multi-cellular eukaryotic organisms, was the first project that attempted to map biological processes to electronic hardware. A newly emerging field that uses models of prokaryotic organisms such as bacteria to create bio-inspired man-made systems is a related but different architecture. Here, we name the artificial electronic systems inspired by these unicellular creatures, 'Unitronics' [6, 7, 8]. The Unitronics system uses two different types of cells; core cells (C-cell), surrounded by peripheral cells (P-cell) around its perimeter (Fig. 3).

Core cells are configured to implement specific functions, as defined by the genes in their configuration register. Peripheral cells on the other hand only manage the input and output information flow, including signal swapping during

test mode. Unitronics adapts a 'see-of-gates' architecture (Fig. 3) similar to that used by commercial FPGAs but partitions the system into prokaryotic islands. Islands are formed by groups of C-cells surrounded by P-cells. Peripheral cells (Fig. 4) of the array provide an interface between the island of C-cells and the outside world. They consist of two flip-flops and a signal controller. They have four bi-directional pins, two of which (P1 and P2) provide communication with the peripheral bus (P-BUS), and the other two (E1 and E2) provide communications with the global bus (G-BUS). Signal directions in E and P are defined by the appropriate configuration bits for the P-Cell. The flip-flops receive their data either from the External (E) or from the Peripheral (P) bus lines, under the control of two multiplexers. External communication can be disabled in order to swap data of P1 and P2. This is accomplished by the two flip-flops; connected in this case as a circular shift register.

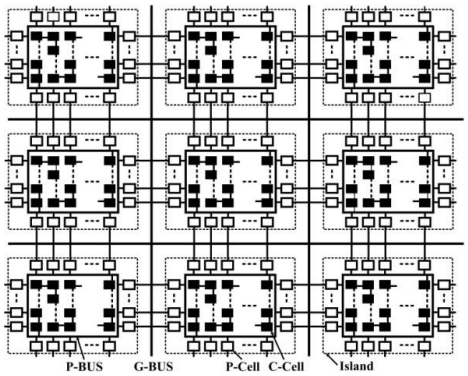

Fig. 3, Schematic diagram of Unitronics

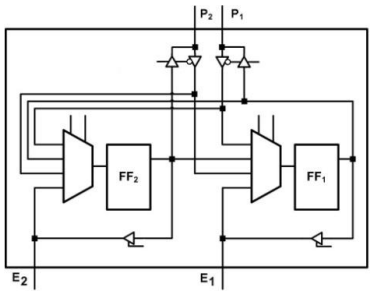

Fig. 4, Peripheral cell, P-Cell

During test mode, data from the P-lines are loaded into the flip-flops are swapped round, and placed back onto the same lines. As a result the lines now have swapped data, as compared with what they had before. Fig. 4 shows only those components of the peripheral cell that provide data switching between P1 and P2 lines.

The array has 2 different types of buses: G-BUS, P-BUS and P-BUS (Peripheral Bus). G-BUS is used for distant communication between C-Cells in different islands via their own P-Cells where signal swapping is also possible.

P-Cells provide flexible connection between any two lines of the G-BUS to any two P-BUS lines. Lines are grouped in pairs, so that once a line is selected as input/output from G-BUS to P-BUS, the second line provides switch over when (e.g. in test mode) required. For self-repair there are additional redundant spare P-Cells.

P-BUS, on entering the array of C-Cells, is divided to C-BUS (Configurable Bus) and L-BUS (Local Bus). They are interconnecting wires, lines and channels, similar to commercial FPGAs. C-BUS provides the required cell to cell interconnect. It is configured by the core cells according to their functional and communicational requirements. Lines of the configurable bus can be grouped, cut, joined and swapped. The bus also supports cell elimination during self-repair if a cell developed a hardware fault. In this case, the faulty cell is killed, its functionality is shifted to the next cell along the configurable bus and all preceding cells are also shifted until a healthy stand-by cell is found. The L-BUS, though can be divided to sub-sections, usually passes through the cells and only makes connection to those with which long distance data communication is required. It is local to the island, and would normally connect to the P-BUS only at the first and the last cell of the island.

C-Cells are the processing and communication elements of the system and as such they provide processing Function (F), signal Routing (R), information storage as Memory (M), and switching as Void (V) tasks. The two slices of the cell can work in tandem and undertake any combination of the above tasks as for instance FF, FR, MV, RM and etc. The detailed architecture of configurable bus is beyond the scope of this paper. The cell's Connection Box (CB) manages how the cell should be connected to the network of other cells in the island. Inputs to the cell's Function Unit (FU) are provided either from the bus via the CB or from the cell's neighbours via dedicated neighbouring connections lines.

FU includes two 2-bit slices. Each slice is supported by the cell's genome, which is essentially an LUT. It can either define the precise function the slices should execute, or can configure them for signal routing. Slice function can either be logical or algebraic. When for example a cell is configured as RF then slice 2 will undertake signal routing, while slice 1 will execute a function on its output. FF set-up enables the cell for a more sophisticated function.

The cell can be used as a memory to implement registers, counters and, in case of a distributed memory, an 8, 16, 24 or 32-bit RAM. It is called a distributed memory because one cell can only provide up to two memory locations. The configuration bit (Ccv) register is not an addressable memory. To allow such functionality a distributed memory feature has been designed. In this case another cell is used as a memory controller. When the cell acts as a "Void" it provides a connection between C-BUS and L-BUS. If a cell is used for M or V the functionality of its slices' is reduced.

In summary the Unitronic architecture, inspired by biological colonies and the circulatory system of a Biofilm, is a network of colonies supported by adequate routing and communication facilities for the cellular array. Both hard and 'soft' entities of the architecture demonstrate biological inspiration. Cells, islands and the circulatory system are the hardware components, and clusters, colonies and biofilms are the 'software' components of the Unitronic system. There is no physical location in the array that can be identified as being a cluster, or colony. Both are 'soft components'

providing immune protection for the system for fault detection and repair. The architecture in Fig. 3 is a substrate where cells, cluster, colonies and biofilms are grown in the islands located in the network of voids and circulatory system.

1.5. Robot Controller

In this example, to demonstrate the self-healing and self-repair capability of Unitronics, the timer part of a movement controller for an e-puck object avoidance robot from EPFL [11] is implemented on a Unitronics array. The Unitronic timer part is synthesised on a Xilinx XUPV5-LX110T development board [12], while the movement part of the controller and the interface between the robot and the Unitronics system is provided by Matlab. Using hardware co-simulation, data from the Unitronics array is transferred to Matlab in a 2-bit data. One bit defines whether a right or left turn is required from the robot, while the other is a fault indicator for the Unitronic system.

The timer is a 16-bit up counter the implementation of which required eight Unitronic cells. Fig. 5 shows the cells' genomes that implement the timer. The slices of all the cells, in this example, are configured as function-function (FF) and define a full adder. In reality the circuit offers a 16-bit full adder, but with inputs set to '0' and carry-in set to '1', it behaves as a 16-bit counter. MSB bit of this counter describes whether robot should turn right or left. Combination of turning right and left makes the robot to move in a figure 8-like manner. Since the genome of every cell is the same, their identical CSV translates into one sv-cluster and their Δg (equalling to zero) into one Δg-cluster. TΔg, and Tcv tag values are chosen arbitrarily as "10" and "11" respectively.

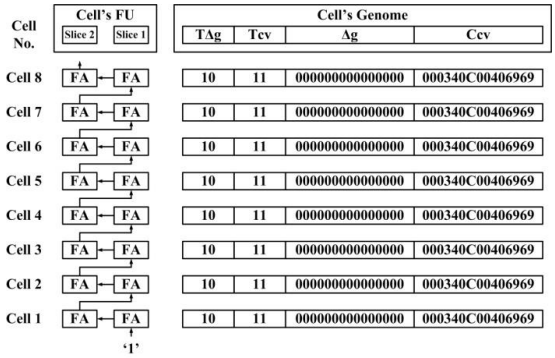

Fig. 5. Unitronic timer implementation (values shown in hex)

Since all cells are located in the same sv-cluster and in the same Δg-cluster, fault recovery is always guaranteed for as long as there is one healthy cell in the system. This example uses the simple algebraic function in Equation 4:

$$Ccv(Tsv, T\Delta g) = Csv(Tsv) + \Delta g(T\Delta g) \quad (4)$$

Since in this example Δg = 0 means that CCV = CSV. Consider a situation when seven out of the 8 cells are faulty and only one functions correctly. If we assume that all tags are correct and cell 5 is the faultless cell then after eliminating the faulty cells the next step is a shift process. With this, if the cells are sequentially placed along the bus, cell1 will assume the position of cell5 and the remaining cells occupy positions cell 9 to cell 15 of the stand-by cells.

The next step is to search in the sv-cluster space and identify the faulty cell's shared value. This is achieved by sending a token that will locate the first faulty cell, in this case cell 15. In order to find the shared value of this cell its Tsv tag is sent to all cells in the cluster. Since only cell 12 is healthy, the tag requests the extraction of its shared value using the re-arranged form (i.e. Csv = Ccv - Δg) of equation 4. This here will coincidentally yield the same as the Ccv value of cell 12 and be released to the bus. All those cells which need the recovery of their shared value and have the same Tsv as cell 15, will receive it. In this case it will affect all cells of the cluster except cell 12. The final step of the repair process is to differentiate it with all the faulty cells' Δg. Since Δg is zero for them all, their configuration vector can now be simultaneously recovered, using equation 4.

In this example cluster identification is trivial due to the repetitive nature of the cell functions required. This in larger digital systems becomes more difficult. These however are typically composed of regular building blocks, i.e., registers, counters, multipliers etc; where this regularity can be exploited to simplify cluster formation.

Another example of a PD controller is shown in Fig. 6. The waveform illustrates the actual behaviour of the hardware (not simulation results!) and the fault recovery process of the controller. The PD controller was also implemented, also as an interim step before VLSI implementation, on a Xilinx XUPV5-LX110T development platform. The controller required 40 Unitronic cells and a 'soft' fault was injected in the genome of cell 3.

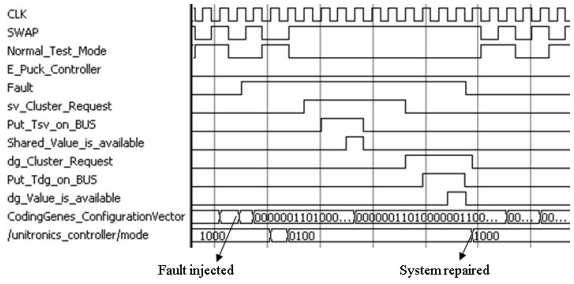

Fig. 6, Implemented robot controller fault recovery

During the operation of the robot controller a fault was inserted into cell3. Fig. 6 shows the fault recovery process of the implemented system:

- 1. Fault is injected at fault injected point into the system.
- 2. The effect of the fault causes the gene to mutate at CodingGenes_ConfigurationVector.
- 3. Simultaneously self-test using input data and control sequence complementation recognises it, identifies the faulty cell and initiates self-repair.
- 4. Self-repair requests the mutated faulty cell's CSV at sv_Cluster_Request. For this TSV at Put_Tsv_on_BUS identifies the cluster and the cells that share the same portion of the configuration vector with the faulty cell.

With the aid of the cluster's cells, CSV is calculated at Shared_Value_is_availabe.

- 5. Recalculation of the faulty cell's corrupted CCV configuration vector also requires its Δg.
- 6. Δg's address TΔg is triggered at Put_dgTag_on_the_BUS in order to locate the same Δg.
- 7. When Δg is also available, using Equation (4) the faulty cell's CCV can be calculated (dg_Value_is_available='1').
- 8. With its recovery, on-line repair of the faulty cell is complete and the recovered correct response result of the cell is now allowed to propagate to its final output.
- 9. Normal system operation (at System repaired) in the next machine cycle resumes as if fault never occurred.

1.6. Conclusion

On-line fault detection and fault repair capability of our Unitronics architecture, based on the bio-inspired prokaryotic model, is demonstrated using an e-puck object avoidance mobile robot. Implementation of the robot required 8 Unitronic cells appropriately interconnected and then mapped onto a Xilinx XUPV5-LX110T development board. The fault tolerance model of the system guarantees that "if similarities and differences between healthy and faulty cells are known then, full recovery of any Unitronic implemented system is possible". The system is able to cope with and repair any number of simultaneously occurring dynamic (SEU) or static (hardware) faults. The amount of fault repair only depends on the number of spare cells the system is equipped with. Its fault repair uses significantly less memory for gene storage and considerably less hardware overall for target system implementation than any previously proposed bio-inspired architecture.

Acknowledgements

This research work is supported by the Engineering and Physical Sciences Research Council of the United Kingdom under Grant Number EP/F062192/1.

References

[1] Garis, H. de. (1993). Evolvable Hardware. The Genetic Programming of Darwin Machines [C].In Proceeding of Artificial Neural Nets and Genetic Algorithms, pages 441-449.

[2] Mange, D. (1996). Embryonics: a new family of coarse-grained FPGA with self-repair and self-reproducing properties. Towards Evolvable Hardware: An evolutionary approach. Springer Verlag. Pages 197-220.

[3] Mange, D. and Sipper, M. and et al. (2000). Towards Robust Integrated Circuits: The Embrynics Approach. Proceedings of the IEEE, vol.88, no.4, pages 516-541.

[4] Barker, W., Halliday. D. M., Thoma. Y., and et al (2007). Fault Tolerance using Dynamic Reconfiguration on the POEtic Tissue, IEEE Transactions on Evolutionary Computation, Vol. 11, No. 5, pages 666-684.

[5] Macias, N. Durbeck, L. Prokopenko, M. (2008). Advances in Applied Self-organizing Systems. Springer.

[6] Samie, M., Dragffy, G. Pipe, T. and et. Al (2009). Prokaryotic Bio-Inspired Model for Embryonics. AHS'09 - NASA/ESA Conference on Adaptive Hardware and Systems, pages 163-170.

[7] Samie, M. Dragffy, G., Pipe, T. and et al (2009). Prokaryotic Bio-Inspired System. AHS'09 - NASA/ESA Conference on Adaptive Hardware and Systems, pages 171-178.

[8] Samie, M., Dragffy, G., Pipe, T. (2010). Bio-Inspired Self-Test for Evolvable Fault Tolerant Hardware Systems. AHS2010 - NASA/ESA Conference on Adaptive Hardware and Systems. pages 325 – 332.

[9] Jacob, F., Brenner, S., Cuzin, F. (1963). The Regulation of DNA Replication in Bacteria. Cold Spring Harbor Symposia Quantitative Biology. pages 329–348.

[10] Kidwell, M. G. (2005). Transposable elements. In ed. T.R. Gregory. The Evolution of the Genome. San Diego: Elsevier. Pages 165–221. ISBN 0-12-301463-8

[11] Mondada, F., Bonani, M., Raemy, X. and et al. (2009). The e-puck, a Robot Designed for Education in Engineering. Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, vol. 1, num. 1, pages 59-65.

[12] XUPV5 - LX110T User manumal, http:// www.xilinx.com/ univ/ xupv5-lx110T-manual.htm

2013-09-27

Unicellular self-healing electronic array

Samie, Mohammad

Elsevier

Mohammad Samie, Gabriel Dragffy, Tony Pipe, Suresh Perinpanayagam. Unicellular self-healing electronic array. 2nd International Through-life Engineering Services Conference, TESConf 2013, Cranfield, 5 November 2013. Procedia CIRP, Volume 11, 2013, Pages 400-405
https://doi.org/10.1016/j.procir.2013.07.009